# Extending an XML environment definition language for spoken dialogue and web-based interfaces


| Pablo A. Haya | Germán Montoro | Xavier Alamán | Rubén Cabello | Javier Martínez |
|---|---|---|---|---|
| EPS-UAM | EPS-UAM | EPS-UAM | EPS-UAM | EPS-UAM |
| Madrid, Spain | Madrid, Spain | Madrid, Spain | Madrid, Spain | Madrid, Spain |
| +34 91 497 22 67 | +34 91 497 22 10 | +34 91 497 22 50 | +34 91 497 22 68 | +34 91 497 22 54 |
| Pablo.Haya @uam.es | German.Montoro @uam.es | Xavier.Alaman @uam.es | Ruben.Cabello @uam.es | Javier.Martinez @uam.es |



**ABSTRACT**

In this work we describe how we employ XML-compliant languages to define and intelligent environment. This language represents the environment, its entities and their relationships. The XML environment definition is transformed in a middleware layer that provides interaction with the environment. Additionally, this XML definition language has been extended to support two different user interfaces. A spoken dialogue interface is created by means of specific linguistic information. GUI interaction information is converted in a web-based interface.



**Keywords**

Interface design, XML, UIDL, intelligent environments, spoken dialogues, web interfaces.


## INTRODUCTION

Within the ubiquitous computing [13] research area it is necessary the study of the design of transparent user interfaces for the interaction with intelligent environments [4]. These interfaces provide new ways of interaction [14], adapt to the users and the environment and offer new challenges to interface designers [11].

Intelligent environment interfaces can range from a GUI mobile-interface (for instance a web-based interface, accessible from a computer or a PDA) to a higher-level interface (such as a spoken dialogue or a gesture-based interface).

Given the dynamic characteristics of intelligent environments, these interfaces have to be easily configurable and adaptable [7, 10] and have to provide standard methods of definition and configuration.

Bearing in mind these conditions we have developed an XML-compliant language that allows to define the characteristics of an intelligent environment. Furthermore, we have added interface information to the language, creating a user interface description language (UIDL) that permits to automatically create a web-based interface and spoken dialogue interface based on the environment information.

Here we present the main ideas of our XML-based language that defines the intelligent environment and these two interfaces. Next sections are organized as follows: first, we give brief overview of user interface definition languages; next, we describe the environment representation through our XML language; after that, we present the definition of the web-based and spoken dialogue interfaces and, finally, we present the conclusions.

## USER INTERFACE DEFINITION LANGUAGES

XML presents as a solution for the standardization of the interoperability between applications. Therefore, new XML-compliant languages are employed to define user interfaces. These are the XML-compliant user interface definition languages (XML-UIDL). They have the advantage of being transparent to different interface technologies and providing a homogeneous resource for heterogeneous types of interaction [1].

According to [12] these XML languages for interface representation must be applicable to any target, any delivery context, personalizable, flexible and extensible. On the other hand, they should separate the interface elements from their presentation. The user interface elements must be explicitly represented and in a format that can be rendered in any delivered context. The presentation information should be provided in an abstract form that is target and delivery-context independent.

Two representative languages are:

- UIML [2], an XML-compliant language which permits creation of user interfaces for any device, any target language and any operating system. It describes the appearance of the user interface, the user interaction with the interface and how it is connected to the application logic.
- XIML [9], an XML-based "interface representation language for universal support of functionality across the entire lifecycle of a user interface: design, development, operation, management, organization, and evaluation".

Other languages are XUL [6], that allows to build easily customizable graphical user interfaces for multiple

platforms. AAIML [15], an XML-based language used to communicate an abstract user interface definition for a service or device to a user's personal device. And XAML [8], the Microsoft XML based language employed for visual interfaces to define a layout of text, images and controls.

### XML ENVIRONMENT DEFINITION

The physical environment is represented in a document, where each environment entity is described using an XML format. Entities are not only formed by the physical devices presented in the environment, but also by software applications, people definition or abstract concepts.

This XML representation also allows to describe the relationships between the environment entities. These relationships define the distribution of the environment (buildings, rooms, etc.), aggregations of people (by workgroups, range, etc.) and dynamic links between entities (the favorite paintings of a person, the output speaker for a music source, etc.).

XML information is processed by a parser and transformed in a middleware layer, which will act as an interaction layer between the user interfaces and the physical environment.

The middleware implementation lies on a global data structure, called blackboard [5]. This blackboard is a model of the world, where all the prominent information related to the environment is stored. The blackboard provides an asynchronous communication mechanism. Senders publish environment information in the blackboard, and receivers can be subscribed to these changes or pull them directly from the blackboard. This mechanism permits a loosely-coupled interaction among senders and receivers, given that it is not needed that either both of them are active at the same time or they know each other. Therefore, the blackboard allows communicating environment changes, finding available devices and revealing if a device has been added or removed.

#### Environment representation

The environment information stored in this blackboard can be viewed as a two-layered structure. On the one hand, a relationship layer has information about the relations between entities. On the other hand, an entity layer stores information about each particular entity.

The relationship layer is a non-directed graph where each node is an entity. Each entity node represents relevant environment information such as physical devices, software applications, occupants or abstract concepts. Arcs between entity nodes denote some kind of relationship (composition, aggregation, association, etc.). For example, the location of a person is modelled as an arc between that person and the room where s/he is located. Given that we employ non-directed arcs, reciprocal relations are also modelled. Therefore, each room has a relationship with every one of its occupants.

Every entity node has assigned a name. This is a unique alphanumerical string. This way, the node name univocally represents the entity. Moreover, entity nodes hold extra information that indicates the entity type (an entity can be a device, a person, a room, etc.).

The entity layer is composed as follows. Every entity has a collection of properties. Entities of the same type inherit a set of common properties, which defines their specific characteristics. Besides, the entities can define new common properties, called parameters, which represent custom information for that entity.

The composition of each environment entity is reflected in the blackboard as a tree structure. The tree root is one of the nodes of the previously described relationship graph. This node has a set of child nodes that defines its properties and parameters.

A property node constitutes an intrinsic and universally accepted feature of the entity. Properties have a name and a value. Thus, two properties that belong to the same entity must have distinct names. Values are leaf nodes that store literal values which can be of type string, integer or real. Besides, the changes on the property values that represent physical variables are reflected in the real world. Thus, when an application or an interface needs to get or to change the physical state of a device, it only has to access to the right node in the graph and get or change its value.

Parameter nodes represent a set of specific features defined by an application or a group of applications, and allow to customize the entity model. Parameters hang of a parameter set node (aka *paramSet* node). Each group of applications can define its own *paramSet* independently of the rest. Parameters, like properties, are name-value pairs. Nevertheless, they can be associated not only to an entity but also to a property. This mechanism provides fine-grain parameterisation.

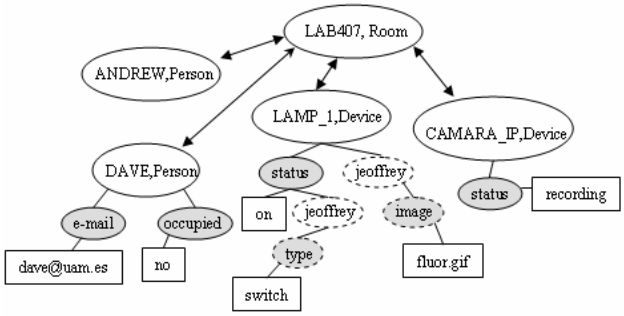


**Figure 1.** Blackboard: entities and their relationships

So, combining these two layers, the resulting blackboard structure can be seen like a graph of entities, where each entity is described as tree of properties and parameters. Figure 1 depicts a schematic blackboard graph. It contains five entity nodes, four property nodes and two *paramSet* nodes with one parameter each. Entities are within a blank circle, with its name and type (for instance, Andrew and person). Double-arrow lines indicate a bidirectional

relationship. Shadowed solid circles represent property nodes (for instance, e-mail), blank dashed circles represent *paramSets* (for instance, Jeoffrey) and shadowed dashed circles represent parameters (for instance, image). Finally, rectangles hold the node value (for instance, dave@uam.es).

This structure allows to organize the environment information using several abstraction levels. The deepest nodes represent more concrete properties, while the upper nodes in the hierarchy reflect structural relationships among entities.

#### Name Space

An entity node can be indexed by its name. Besides, any node can be located, starting from any entity node and following the relationship path. This is called the node path. It is composed by a list of tokens separated by the slash character. Their order is determined as follows: the first token of the path is the word "name", the second one must be the entity name and the next tokens come as the result of concatenating the names of all the intermediate nodes until the target node. For instance, in the example showed in the figure 1, the lamp_1 status path is */name/lamp_1/status*.

In addition, wildcards can be used to substitute one or several tokens. This allows referencing several nodes at the same time. For example, based on the Figure 1, */name/dave/**  references all the properties, *paramSets* and related entities of the entity *Dave*. As a result it gets the following list: the e-mail and busy property nodes and the Lab_407 entity node.

Another two naming mechanisms are provided to improve the use of wildcards:

- Predefined hierarchy. This mechanism restricts the nodes that compose a path. It specifies how to go through the graph. To do this, each hierarchy defines a sequence of types of entities. For example, the first type of entity must be a room, the second one a device, etc… Therefore, when a wildcard is used, only the nodes that match with the expected type will be substituted. These hierarchies are called predefined because they are hard-wired. Following with the example of the figure 1, the path */roomdevice/lab407/*/props/status* is interpreted as follows: the initial token identifies the hierarchy *roomdevice*. This hierarchy establishes that the first type of entity must be a room followed by a device. The other nodes remain unrestricted. Therefore, this path references the value of the status of all the devices located in lab407.
- Typed hierarchy. This is a particular case of the previous mechanism. By default, there will be as many hierarchies as types of entities. The initial token of these hierarchies is the type of entity. For example, in the figure 1 there are three default hierarchies: person, room and device, so that */person/*/mail* retrieves the e-mails from everybody.

```
<entity name="id" type="type">
    <property name="name">value</property>
    <property name="name">value
      <paramSet name="name">
        <param name="name">value</param>
        <param name="name">value</param>
          .....
      </paramSet>
    </property>
    .....
    <paramSet name="name">
      <param name="name">value</param>
      <param name="name">value</param>
       .....
    </paramSet>
    <paramSet name="name">
      <param name="name">value</param>
      <param name="name">value</param>
       .....
    </paramSet>
    ......
    <entity name="name"/>
    …..
</entity>
```

**Figure 2.** XML template of an entity

#### Interaction with the blackboard

Interfaces do not interact directly with the environment physical entities but they only have access to the middleware information. So, the implementation details of an entity are hidden to the applications and they only have to use the same standard communication rules for any entity of the environment.

The middleware provides a set of operations that allows to: retrieve the information stored in blackboard, make changes on the values of the properties and add or remove an entity or a relationship. To access or to change the blackboard information, applications and interfaces employ a simple communication mechanism through the HTTP protocol, by means of XML-compliant messages.

Figure 2 shows an XML representation of a generic entity obtained from the blackboard.

Thereby, the initial backboard structure can be generated from a set of XML files that store the environment configuration.

As we have seen in this section it is simple and standard to describe the environment, to retrieve the state of its entities

or change it. The XML-compliant definition language serves as a standard tool to specify the characteristics of the environment. Once created, to get or to change the physical state of the environment or to add or remove new entities is also possible by means of standard instructions.

### XML INTERFACE DEFINITION

Besides the entity property definition, employed to build the middleware layer, the entities have associated other XML information employed to automatically build diverse user interfaces.

Currently, our XML-compliant environment definition language supports the automatic construction of two different user interfaces: a spoken dialogue interface and a web-based interface.

#### Spoken dialogue interface

Spoken interaction become necessary for an intuitive communication between users and intelligent environments [3]. Considering this, we have added new XML dialogue tags to the environment description, in order to support the automatic creation of a Spanish dialogue interface.

Dialogues are associated to each entity, so that when a new entity appears in the environment a new dialogue allows the users to interact with that entity. If the entity is not part of the environment, the dialogue will not be available.

Each dialogue entity depends on the type of entity, so the entities of the same type will inherit the same kind of possible interactions. Entity dialogues can be customized for each entity, in order to distinguish between them. A supervisor is in charge of managing the dialogue interactions, resolving conflicts when there are several entities of the same type, among many others.

```
<class name="fluorescent">
<property name=”Status”>
<paramSet name=”dialogue”>
 <paramSet name=“sentence“>
  <param name=“verbPart”>turn_on switch_on</param>
  <param name=“objectPart”>light</param>
  <param name=“modifierPart”> </param>
  <param name=“locationPart”>ceiling above</param>
  <param name=“indirectObjectPart”></param>
 </paramSet>
 <paramSet name=“sentence” >
  <param name=“verbPart”>turn_on </param>
  <param name=“objectPart”>fluorescent</param>
  <param name=“modifierPart”> </param>
  <param name=“locationPart”> </param>
  <param name=“indirectObjectPart”></param>
 </paramSet>
</paramSet>
</property>
</class>
```

**Figure 3.** Linguistic information for an entity definition

Each entity must have associated all the possible ways a user can interact with it. For this we have defined an initial set of linguistic parts, which tries to cover all the possible interactions between the user and the entity. This set is formed by:

- A verb part, which corresponds with the action that the user wants to perform with the entity.
- An object part, related with the name that the user gives to the entity.
- An indirect object part, the person who receives the action.
- A modifier part, the kind of object part entity.
- A location part, which informs of the location of the entity in the environment.

The last two parts permit to distinguish between several entities of the same type. These linguistics parts allow the use of synonyms and there can be as many sets of parts as is necessary for each entity. The figure 3 shows the definition of two different sets of linguistic parts for one entity of type fluorescent. Translating the case from Spanish, it is considered that a user could utter sentences of the type: “please, could you switch on the ceiling light” but not of the type “please, could you switch on the fluorescent” (for fluorescents, users only employ the verb turn on). Besides, some parts contain synonyms (turn on and switch on, or ceiling and above).

```
<entity name=”Lamp_1” type="fluorescent">
 <property name=”status”>
  <paramSet name=”dialogue”>
   <paramSet name=“sentence” >
    <param name=“modifierPart”>main</param>
   </paramSet>
  </paramSet>
 <property>
</entity>
```

**Figure 4.** Customized entity instance

To create an entity based on a defined type it is only necessary to create an instance of the entity type. This entity instance inherits all the entity type definition properties, including the linguistic information. In many cases, it will not be necessary to customize this linguistic information, and to declare the entity will be enough to automatically add its dialogue interactions to the interface.

In other cases, the entity instance can be customized to adapt to the environment specific characteristics or to distinguish it form other entities of the same type. Figure 4 shows an entity instance customized for a specific environment.

Additionally, the entity type definition also has to declare:

- A grammar template, which serves as the skeleton to define the recognition grammar.

A grammar template has a set of common rules and empty linguistic parts (marked as nil). The nil marks can be filled in with the linguistic parts provided by the previous definition. Figure 5 shows a simplified section of an action grammar template for an imperative sentence. Besides, it also supports noun sentences, subjunctive sentences (in present, past, singular and plural) and interrogative sentences.

```
<imperative sentence> = <imperative verb> [<noun>];
<imperative verb> =   <imperative informal verb>
                    | <imperative formal verb>
                    | <infinitive verb>;
<imperative informal verb> = nil;
<imperative formal verb> = nil;
<infinitive verb> = nil;
```

**Figure 5.** Section of a grammar template

Every word in the set of linguistic parts is sent to a morphological analyzer. This gets its part of speech information and, based on it, retrieves its different forms. Then it adds each word form to the right grammar rule. For instance, based on the exampled showed in the figure 3, the morphological analyzer gets that *turn on* is verb, so that it gets all the possible declinations for that verb (in Spanish, verb declinations change for each mode, tense, number and person). Then, it would add the right forms to the rules <*imperative informal verb*> and <*imperative formal verb*>, among many others.

This process is repeated with each one of the linguistic parts of an entity type, taking into consideration if the word is a noun, a verb, an adjective, etc.

Grammar templates employ fixed rules that not only combine the added words in a proper way but also allow to employ more general and natural utterances, avoiding to use commands. These sentences try to cover the whole corpus of possibilities that a person employs to address to the entity.

The entity designer can use any of the grammar templates available. If a designer needed to define a new kind of grammar template, s/he could employ and declare any new, as well as combine them as necessary. S/he only needs to keep the name for the rules that will be filled in with the entity linguistic parts, this is, rules of the kind <infinitive verb>, <singular male noun>, etc. S/he only has to declare the rules that correspond with linguistic parts that are necessary for the interaction, avoiding to declare those that are not needed.

And finally, it is necessary that it defines a pointer to two different methods:

- An action method, which receives the action requested by the user (the verb part) and performs an action with the entity. To do this task it serves of the middleware layer.
- A state method, which also receives the verb part and returns if the current entity state is the same or different to the user requested state. Again, it also serves of the middleware layer.

The action method is employed to execute the environment physical action requested by the user. It only has to be implemented once by the entity type designer and the entity instances automatically will inherit this method.

The state method is utilized in the interaction process to determinate if the entity instance has to be processed. In the case that the entity has the same state as the user requested state, the dialogue interaction does not need to consider that entity and can continue processing other entities with a different state. Again, this method only has to be defined once by the entity type designer. The interface definition process will automatically inherit this method for every entity of the same type.

Both methods employ the middleware layer to communicate with the physical environment. To do this, they only have to specify the entity property that they want to interact with, if they want to get or set a value for this property and, in the last case, the value that they want to set. As it was explained above, this communication follows a standard process through the HTTP protocol.

Therefore, to declare a new entity instance it is only necessary to define its type and give it a name. The same declaration that is employed to define that a new entity is part of the environment is used to adjunct its dialogue capabilities to the spoken dialogue interface. Only in some cases this declaration must be customized to adapt its linguistic information to the current environment characteristics and this adaptation is usually based on adding new words to the linguistic parts, this is, new customized ways of interaction.

#### Web-based interface

We have developed a web based interface to control environment's devices and appliances. This interface is called Jeoffrey. It is a custom and partial view of the environment information stored in the blackboard. Jeoffrey is programmed to be used in a home environment.

The blackboard contains generic information regarding the number of rooms and the entities that it hosts. Each entity is represented in the blackboard. Its representation includes

the properties required to interact with it. Additionally, new specific information has been added in order to create the Jeoffrey interface. It is composed by three parts structured hierarchically:

- The top level is a stand-alone list box containing the rooms of the house. When the user selects a room, a new window will pop up.
- This new window shows a room map that includes the location of the furniture and entities. The map layout is composed overlapping a fixed background image with each device representation image. Every time the interface is loaded, the map is generated using the blackboard information.
- Finally, a custom control panel is showed when a user clicks on an entity, allowing to interact with it.

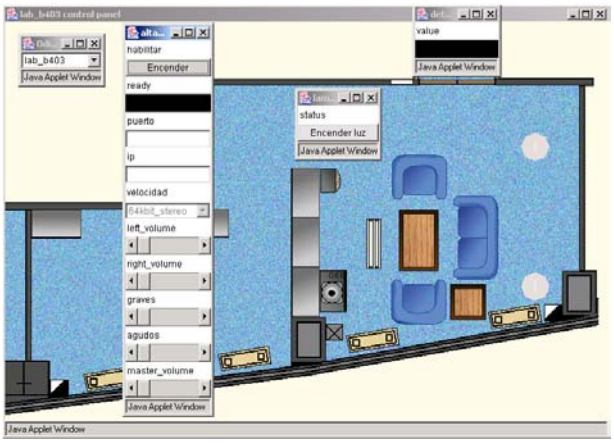


**Figure 6.** Jeoffrey's user interface.

Figure 6 shows a Jeoffrey user interface screenshot. The most left window is the root list box. The background window corresponds to the map that appears when a room is selected. Finally, the other three windows correspond to invoked entity control panels.

Jeoffrey gathers the information stored in the blackboard to dynamically render the user interface. The blackboard graph includes an entity node for each room and for each entity. A relationship between a room and an entity reflects that the entity is located in that room. This way, Jeoffrey can easily ask for all the rooms and, for each of them, which entities are inside.

Each entity includes several Jeoffrey's parameters that help to render its graphical interface. Figure 7 illustrates the Jeoffrey interface information of a fluorescent XML instance. Bold font is used to highlight the Jeoffrey's parameters. There are two *paramSets*. The first one is associated to the entity and contains three parameters. The image parameter defines its corresponding image file. The x and y parameters are the coordinates where this image will be drawn. The second *paramSet* is associated to the status property and defines its related widget.

As we have mentioned above, entity interaction is managed by a custom control panel composed of widgets. This panel is customized depending on the entity properties. Each property is rendered into a widget that allows interacting with the entity property. There are five different generic widgets: text areas, switches, sliders, list boxes or alarms. Text areas permit changing the value of a string. Switches act as a toggle button associated to on-off properties. Sliders correspond to properties that take a value from an interval. List boxes define a list of possible values where the user can choose one. Finally, alarms are colored labels that change its color depending on the value of the property.

```
<entity name="Lamp_1" type="fluorescent">
 <property name="Status“>
   <paramSet name="jeoffrey”>
    <param name="type">switch</param>
    <param name="text_off">Turn on</param>
    <param name="text_on">Turn off</param>
    <param name="cmd_on">0</param>
    <param name=”cmd_off”>1</param>
    <param name="color_on">0x00FF00</param>
  </paramSet>
 </property>
 <paramSet name="jeoffrey">
  <param name="image">reflectante.gif</param>
  <param name="x">460</param>
  <param name="y">247</param>
 </paramSet>
</entity>
```

**Figure 7.** XML entity representation

As figure 7 shows, the fluorescent called Lamp_1 has only a status property. This property is associated with a switch widget. Besides, several switch parameters defining presentation features are established. These features are:

- The button text: this text changes depending on the state of the property. The "*text_off*" parameter is displayed when the light is off whereas the "*text_on*" parameter is showed when the light is on.
- The button color: by default the color is gray when the light is off. When the light is on, the color is defined by the "*color_on*" parameter.

Figure 8 illustrates the rendered control panel for a florescent and the image painted on the map.

Finally, the "*cmd_off*" and "*cmd_on*" parameters define the value of the status property that will be set when the button is pressed.


**Figure 8**. User interface for a fluorescent

When a user clicks on the picture of an entity, Jeoffrey reads the descriptions of its properties from the blackboard, translates the properties to widgets and generates a custom control panel. If the entity has more than one property, the control panel will be composed by the aggregation of the widgets corresponding to each property.

Jeoffrey employs the blackboard as a proxy to interact with the physical entities, for instance, to change the speaker volume, switch on the lights, etc., and to receive the changes occurred in the environment. Jeoffrey is subscribed to every event. All the changes in the state of an entity are reflected in the user interface. For instance, if a property has associated a widget alarm, when its value changes, the blackboard will notify this to Jeoffrey and it will modify the color of the alarm widget.

## CONCLUSIONS

We have presented a graph model that allows to represent the entities of an intelligent environment and their relationships. This model is created using a XML-compliant language, and it is stored in a global data structure, called blackboard. A blackboard middleware provides a set of operations to interact with the graph model. An application can add and remove entities, retrieve and modify their state, and subscribe to the changes done by other applications.

Two user interfaces have been developed to interact with the environment. These interfaces are created by means of an extension of the environment XML model. The first extended language automatically creates a customized spoken dialogue interface. This language adds linguistic information to the XML model. The second one dynamically builds a web based interface. Again, new XML tags allow to specify GUI information.

The middleware and the interfaces have been developed in a real environment. It is composed of several devices, including different types of lights, sensors, a door opening mechanism, an FM tuner, etc. Both interfaces provide real interaction with these devices.

## ACKNOWLEDGMENTS

This work has been sponsored by the Spanish Ministry of Science and Technology, project number TIC2000-0464.

## REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., and Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language. In *Proceedings of the Eighth International WWW Conference*, Toronto, Canada, 1999.
2. Ali, M.A., Pérez-Quiñones, M.A., Abrams, M., and Shell, E. Building Multi-Platform User Interfaces with UIML. In *Proceedings of CADUI*, 2002.
3. Brumitt, B., and Cadiz, JJ. "Let there be light!" Comparing interfaces for homes of the future. In *Proceedings of INTERACT '01*, 375–382, 2001.
4. Coen, M.H. Design Principles for Intelligent Environments. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, Palo Alto, California, 1998.
5. Engelmore, R., and Morgan, T. Blackboard Systems. *Addison-Wesley*, 1988
6. McFarlane, N. Rapid Application Development with Mozilla. Bruce Perens' Open Source Series. *Prentice Hall*, 2003
7. Paternò, F., and Santoro, C. One Model, Many Interfaces. In *Proceedings of CADUI*, 2002.
8. Petzold, C. Create Real Apps Using New Code and Markup Model. *MSDN Magazine*, January 2004.
9. Puerta, A. and Eisenstein, J. XIML: A Universal Language for User Interfaces. *White paper*. Available at http://www.ximl.org/Docs.asp. 2001.
10. Rayner, M., Lewin, I., Gorrell, G., and Boye, J. Plug and Play Speech Understanding. *2nd SIGdial Workshop on Discourse and Dialogue*, September 2001.
11. Shafer, S., Brumitt, B., and Cadiz, JJ. Interaction Issues in Context-Aware Intelligent Environments. *Human-Computer Interaction*, 16, 363-378, 2001.
12. Trewin, S., Zimmermann, G., and Vanderheiden, G. Abstract user interface representations: How well do they support universal access?. In *Proceedings of the 2nd ACM International Conference on Universal Usability*, Vancouver, Canada, 2003.
13. Weiser, M. The computer of the 21st century. *Scientific American*, 265, 3, 66-75, 1991.
14. Weiser, M. The world is not a desktop. *ACM Interactions*, 1, 1, 7-8, 1994.
15. Zimmermann, G., Vanderheiden, G., and Gilman, A. Universal Remote Console Prototyping of an Emerging XML Based Alternate User Interface Access Standard. In *Proceedings of the Eleventh International WWW Conference*, Hawaii, 2002.